UNITED STATES PATENT AND TRADEMARK OFFICE MN

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/726,902	12/03/2003	Mitchell Alsup	5500-88700	4177

53806 7590 07/18/2007

MEYERTONS, HOOD, KIVLIN, KOWERT & GOETZEL (AMD)
P.O. BOX 398
AUSTIN, TX 78767-0398

EXAMINER	
FENNEMA, ROBERT E	
ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
07/18/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/726,902	ALSUP ET AL.	
	Examiner	Art Unit	
	Robert E. Fennema	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *09 May 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-35* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-35* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *5/8/2007*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-35 have been considered. Claims 1, 6, 8, 14, 19, 21, 27, 28, 29, 30, 31, 33, 34, and 35 have been amended as per Applicant's request.

Claim Rejections - 35 USC § 102

2. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

3. Claims 1-2,11-15, and 24-26 are rejected under 35 U.S.C. 102(b) as being anticipated by Rotenberg et al. (herein Rotenberg).

4. As per Claim 1, Rotenberg teaches: A microprocessor (Abstract), comprising:

an instruction cache configured to store instructions (Section 2.1, first paragraph);

a branch prediction unit (Section 2.1, first paragraph);

a trace cache configured to store a plurality of traces of instructions (Section 2.2, second paragraph); and

a prefetch unit coupled to the instruction cache, the branch prediction unit, and the trace cache (Figure 4, and Section 2.2. The Instruction latch appears to fill the role of a prefetch unit);

wherein the prefetch unit is configured to fetch instructions from the instruction cache until the branch prediction unit outputs a predicted target address (Section 2.2, where it is said that "on a trace cache miss, fetching proceeds normally from the instruction cache); and

wherein the prefetch unit is configured to check the trace cache for a match for the predicted target address in response to the branch prediction unit outputting the predicted target address (inherent in Rotenberg);

wherein the prefetch unit is configured to not check the trace cache for a match until the branch prediction unit outputs the predicted target address (inherent in Rotenberg); and

wherein in response to the prefetch unit identifying a match for the predicted target address in the trace cache, the prefetch unit is configured to fetch one or more of the plurality of traces from the trace cache (Section 2.2, where it is said that "on a trace cache hit, an entire trace of instructions is fed into the instruction latch, bypassing the instruction cache).

5. As per Claim 2, Rotenberg teaches: The microprocessor of claim 1, wherein the branch prediction unit is configured to output the predicted target address in response to a prediction that a branch will be taken (Section 2.1, which discloses "They (the BTB banks) serve the role of detecting branches in the instructions currently being fetched and providing their target addresses, in time for the next fetch cycle. A few paragraphs

further down, it is stated that this happens "if there is a predicted taken branch").

6. As per Claim 11, Rotenberg teaches: The microprocessor of claim 1, wherein each of the plurality of traces comprises partially-decoded instructions (it is inherent that a trace comprises a partially-decoded instruction, otherwise the necessary control information as show in section 2.2 would not be available).

7. As per Claim 12, Rotenberg teaches: The microprocessor of claim 1, wherein each of the plurality of traces is associated with a tag comprising the address of an earliest instruction, in program order, stored within that trace (Section 2.2, where the tag identifies the starting address of the trace).

8. As per Claim 13, Rotenberg teaches: The microprocessor of claim 1, wherein each of the plurality of traces is associated with a flow control field comprising a label for an instruction to which control will pass for each branch operation comprised in that trace (Section 2.2, the "trace target address" and "trace fall-through address" are both labels which describe where control will flow based on each branch operation, based on the prediction (which is part of another label, "branch flags")).

9. As per Claim 14, Rotenberg teaches: A computer system, comprising:

a system memory (inherent if the system has an instruction cache); and

a microprocessor coupled to the system memory (also inherent in Rotenberg's invention), comprising:

an instruction cache configured to store instructions (Section 2.1, first paragraph);

a branch prediction unit (Section 2.1, first paragraph);

a trace cache configured to store a plurality of traces of instructions (Section 2.2); and

a prefetch unit coupled to the instruction cache, the branch prediction unit, and the trace cache (Figure 4, and Section 2.2. The Instruction latch appears to fill the role of a prefetch unit);

wherein the prefetch unit is configured to fetch instructions from the instruction cache until the branch prediction unit outputs a predicted target address (Section 2.2, where it is said that "on a trace cache miss, fetching proceeds normally from the instruction cache);

wherein the prefetch unit is configured to check the trace cache for a match for the predicted target address in response to the branch prediction unit outputting the predicted target address (inherent in Rotenberg);

wherein the prefetch unit is configured to not check the trace cache for a match until the branch prediction unit outputs the predicted target address (inherent in Rotenberg); and

wherein in response to the prefetch unit identifying a match for the predicted target address in the trace cache, the prefetch unit is configured to fetch one or more of

the plurality of traces from the trace cache (Section 2.2, where it is said that "on a trace cache hit, an entire trace of instructions is fed into the instruction latch, bypassing the instruction cache).

10. As per Claim 15, Rotenberg teaches: The computer system of claim 14, wherein the branch prediction unit is configured to output the predicted target address in response to a prediction that a branch will be taken (Section 2.1, which discloses "They (the BTB banks) serve the role of detecting branches in the instructions currently being fetched and providing their target addresses, in time for the next fetch cycle. A few paragraphs further down, it is stated that this happens "if there is a predicted taken branch").

11. As per Claim 24, Rotenberg teaches: The computer system of claim 14, wherein each of the plurality of traces comprises partially-decoded instructions (it is inherent that a trace comprises a partially-decoded instruction, otherwise the necessary control information as show in section 2.2 would not be available).

12. As per Claim 25, Rotenberg teaches: The computer system of claim 14, wherein each of the plurality of traces is associated with a tag comprising the address of an earliest instruction, in program order, stored within that trace (Section 2.2, where the tag identifies the starting address of the trace).

13. As per Claim 26, Rotenberg teaches: The computer system of claim 14, wherein each of the plurality of traces is associated with a flow control field comprising a label for an instruction to which control will pass for each branch operation comprised in that trace (Section 2.2, the "trace target address" and "trace fall-through address" are both labels which describe where control will flow based on each branch operation, based on the prediction (which is part of another label, "branch flags")).

Claim Rejections - 35 USC § 103

14. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

15. Claims 3, 16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, in view of Patterson et al. (herein Patterson).

16. As per Claim 3, Rotenberg teaches the microprocessor of claim 1, but fails to teach: wherein the branch prediction unit is configured to output the predicted target address in response to detection of a branch misprediction. However, Patterson teaches that in order to reduce the penalty for a misprediction, you can fetch both the taken and not taken instructions, and put them in the BTB, which would then be able to immediately output the correct path on a misprediction without having to fetch it (Pages 276-277). While it would increase the cost of the system, the advantage is a smaller

misprediction penalty, which may worth the cost, depending on the needs of the system. Therefore, one of ordinary skill in the art at the time the invention was made would have stored both the taken and not taken branch paths in the BTB in order to reduce the misprediction penalty, and thus increasing performance.

17. As per Claim 16, Rotenberg teaches the computer system of claim 14, but fails to teach: wherein the branch prediction unit is configured to output the predicted target address in response to detection of a branch misprediction. However, Patterson teaches that in order to reduce the penalty for a misprediction, you can fetch both the taken and not taken instructions, and put them in the BTB, which would then be able to immediately output the correct path on a misprediction without having to fetch it (Pages 276-277). While it would increase the cost of the system, the advantage is a smaller misprediction penalty, which may worth the cost, depending on the needs of the system. Therefore, one of ordinary skill in the art at the time the invention was made would have stored both the taken and not taken branch paths in the BTB in order to reduce the misprediction penalty, and thus increasing performance.

18. Claims 4, 10, 17, 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, in view of Braught, further in view of Xia.

19. As per Claim 4, Rotenberg teaches: The microprocessor of claim 1, further comprising a trace generator (it is necessary for Rotenberg's invention to have a trace generator in order to create traces), but fails to teach:

wherein the trace generator is configured to begin a trace with an instruction corresponding to a label boundary.

Rotenberg teaches starting a trace on a trace cache miss, which then causes the line-fill buffer to begin filling (Section 2.2, fourth paragraph). Rotenberg does not specifically teach that an instruction must be a branch to access the cache (despite requiring branch prediction bits). However, Xia teaches that an advantage of starting a trace only at predictable branch instructions (Section 5.5, first scheme) is that less cache space is required, in direct contrast to Rotenberg, which teaches tracing on all instructions, requiring significantly more space (second scheme). Given this advantage, one of ordinary skill in the art at the time the invention was made would have been motivated by the advantage of needing less space for the trace cache to implement traces only on branches.

However, this combination still fails to teach starting a trace on a label boundary. Braught teaches that in Assembly Language, most branches operate on labels, which the machine can only interpret when converted to an address (Page 3). In Braughts example on page 3, all branches are labels, making every branch a label boundary. When combined with Rotenberg's invention, this would mean that a trace would be generated with an instruction corresponding to a label boundary, as all the branches would go to labels, and even an address may be interpreted as a label. Given this

knowledge, it would have been obvious to one of ordinary skill in the art that Rotenberg's invention begins traces when it encounters an instruction with a label boundary, as it only begins on branch instructions, which branch to labels.

20. As per Claim 10, Rotenberg teaches: The microprocessor of claim 4, wherein the trace generator is configured to generate traces in response to instructions being decoded (Section 2.2, the trace can not be completed (written into the cache) until the trace target addresses are calculated, which requires the instructions to be decoded).

21. As per Claim 17, Rotenberg teaches: The computer system of claim 14, further comprising a trace generator (it is necessary for Rotenberg's invention to have a trace generator in order to create traces), but fails to teach:

wherein the trace generator is configured to begin a trace with an instruction corresponding to a label boundary.

Rotenberg teaches starting a trace on a trace cache miss, which then causes the line-fill buffer to begin filling (Section 2.2, fourth paragraph). Rotenberg does not specifically teach that an instruction must be a branch to access the cache (despite requiring branch prediction bits). However, Xia teaches that an advantage of starting a trace only at predictable branch instructions (Section 5.5, first scheme) is that less cache space is required, in direct contrast to Rotenberg, which teaches tracing on all instructions, requiring significantly more space (second scheme). Given this advantage, one of ordinary skill in the art at the time the invention was made would have been

motivated by the advantage of needing less space for the trace cache to implement traces only on branches.

However, this combination still fails to teach starting a trace on a label boundary. Braught teaches that in Assembly Language, most branches operate on labels, which the machine can only interpret when converted to an address (Page 3). In Braughts example on page 3, all branches are labels, making every branch a label boundary. When combined with Rotenberg's invention, this would mean that a trace would be generated with an instruction corresponding to a label boundary, as all the branches would go to labels, and even an address may be interpreted as a label. Given this knowledge, it would have been obvious to one of ordinary skill in the art that Rotenberg's invention begins traces when it encounters an instruction with a label boundary, as it only begins on branch instructions, which branch to labels.

22. As per Claim 23, Rotenberg teaches: The computer system of claim 17, wherein the trace generator is configured to generate traces in response to instructions being decoded (Section 2.2, the trace can not be completed (written into the cache) until the trace target addresses are calculated, which requires the instructions to be decoded).

23. Claims 5-8 and 18-21 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, Xia and Braught, further in view of Lange et al. (USPN 3,896,419, herein Lange).

24. As per Claim 5, Rotenberg, Xia and Braught teach the microprocessor of claim 4, but fail to teach:

wherein the trace generator is configured to check the trace cache for a duplicate copy of the trace that the trace generator is constructing.

Rotenberg teaches a standard trace cache, with the potential for some duplication, when considering the alternative embodiment of path associativity disclosed in Section 2.3. And while Rotenberg does not discuss the issue of duplication, he does teach of disadvantages which occur when a trace cache miss occurs while servicing a previous trace cache miss, and that this situation needs to be avoided, otherwise features which increase cost or decrease performance must be implemented, and further teaches the disadvantages of a useless trace displacing a useful trace (Section 2.3, judicious trace selection). Therefore, Rotenberg teaches a system in which allows tracing of potential duplicate traces, and also a system which requires action when a miss occurs while servicing a miss. Lange teaches a system in which a cache is checked for a value while accessing memory, and if the identical value is found in the cache, the fetch to memory is aborted, freeing the memory to be used by a different operation (preventing a blocking load, Column 9, Lines 60-65) This is analogous to Rotenberg in the sense that Lange provides a motivation to look for duplicates in a cache to prevent a large delay and allow another operation to continue (analogous to finding a solution to what to do when encountering a miss while servicing a miss) by implementing a solution of searching for the duplicate, and not performing a long-latency operation on the duplicate value since it is not required (if the previous trace

appears to be a duplicate of what is in the trace, there is no need to continue tracing, and the duplicate value can be discarded as taught by Lange (there is no need to perform an action on a duplicate value), preventing the line-fill buffer from blocking new traces, a long-latency event).

In an alternative viewpoint disclosed above, Rotenberg also teaches why it is disadvantageous to displace a useful trace with a useless trace (under the interpretation that a duplicate trace being stored in the trace cache is useless, as the original can always be accessed in its place), thus, with the potential for duplicate traces to exist with path associativity in Rotenberg's alternative embodiments, Rotenberg indicates in his judicial trace selection that storing a duplicate trace would be at best useless, and at worst displace a useful trace that may be used, providing a motivation to prevent these duplicate traces from entering the trace cache from the line fill buffer, further providing a motivation to find some method to handle these cases, which Lange provides, by simply discarding the duplicate value.

Given these advantages in both of these situations, one of ordinary skill in the art at the time the invention was made would have been motivated to combine the teachings of Lange's checking for duplicates and discarding of the duplicates to Rotenberg, in order to solve either or both of the problems of getting a miss while servicing a miss, and preventing eviction of a useful trace by a duplicate trace.

25. As per Claim 6, Lange teaches: The microprocessor of claim 5, wherein in response to the trace generator identifying a duplicate copy of the trace, the trace

generator is configured to discard the trace under construction (Column 5, Lines 5-10).

26. As per Claim 7, Rotenberg teaches: The microprocessor of claim 5, wherein in response to the trace generator identifying an entry corresponding to a duplicate copy of the trace, the trace generator is configured to check the trace cache for an entry corresponding to a next trace to be generated (Section 2.2. The trade cache is checked every time there is a potential new trace, so when one trace is found and discarded, the next potential new trace will cause the trace cache to be checked again).

27. As per Claim 8, Lange teaches: The microprocessor of claim 7, wherein in response to the trace generator identifying a trace entry corresponding to the next trace to be generated, the trace generator is configured to discard the trace under construction (Column 5, Lines 5-10).

28. As per Claim 18, Rotenberg, Xia and Braught teach the computer system of claim 17, but fail to teach:

wherein the trace generator is configured to check the trace cache for a duplicate copy of the trace that the trace generator is constructing.

Rotenberg teaches a standard trace cache, with the potential for some duplication, when considering the alternative embodiment of path associativity disclosed in Section 2.3. And while Rotenberg does not discuss the issue of duplication, he does teach of disadvantages which occur when a trace cache miss occurs while servicing a

previous trace cache miss, and that this situation needs to be avoided, otherwise features which increase cost or decrease performance must be implemented, and further teaches the disadvantages of a useless trace displacing a useful trace (Section 2.3, judicious trace selection). Therefore, Rotenberg teaches a system in which allows tracing of potential duplicate traces, and also a system which requires action when a miss occurs while servicing a miss. Lange teaches a system in which a cache is checked for a value while accessing memory, and if the identical value is found in the cache, the fetch to memory is aborted, freeing the memory to be used by a different operation (preventing a blocking load, Column 9, Lines 60-65) This is analogous to Rotenberg in the sense that Lange provides a motivation to look for duplicates in a cache to prevent a large delay and allow another operation to continue (analogous to finding a solution to what to do when encountering a miss while servicing a miss) by implementing a solution of searching for the duplicate, and not performing a long-latency operation on the duplicate value since it is not required (if the previous trace appears to be a duplicate of what is in the trace, there is no need to continue tracing, and the duplicate value can be discarded as taught by Lange (there is no need to perform an action on a duplicate value), preventing the line-fill buffer from blocking new traces, a long-latency event).

In an alternative viewpoint disclosed above, Rotenberg also teaches why it is disadvantageous to displace a useful trace with a useless trace (under the interpretation that a duplicate trace being stored in the trace cache is useless, as the original can always be accessed in its place), thus, with the potential for duplicate traces to exist

with path associativity in Rotenberg's alternative embodiments, Rotenberg indicates in his judicial trace selection that storing a duplicate trace would be at best useless, and at worst displace a useful trace that may be used, providing a motivation to prevent these duplicate traces from entering the trace cache from the line fill buffer, further providing a motivation to find some method to handle these cases, which Lange provides, by simply discarding the duplicate value.

Given these advantages in both of these situations, one of ordinary skill in the art at the time the invention was made would have been motivated to combine the teachings of Lange's checking for duplicates and discarding of the duplicates to Rotenberg, in order to solve either or both of the problems of getting a miss while servicing a miss, and preventing eviction of a useful trace by a duplicate trace.

29. As per Claim 19, Lange teaches: The computer system of claim 18, wherein in response to the trace generator identifying a duplicate copy of the trace, the trace generator is configured to discard the trace under construction (Column 5, Lines 5-10).

30. As per Claim 20, Rotenberg teaches: The computer system of claim 18, wherein in response to the trace generator identifying an entry corresponding to a duplicate copy of the trace, the trace generator is configured to check the trace cache for an entry corresponding to a next trace to be generated (Section 2.2. The trade cache is checked every time there is a potential new trace, so when one trace is found and discarded, the

next potential new trace will cause the trace cache to be checked again).

31. As per Claim 21, Lange teaches: The computer system of claim 20, wherein in response to the trace generator identifying a trace entry corresponding to the next trace to be generated, the trace generator is configured to discard the trace under construction (Column 5, Lines 5-10).

32. Claims 9 and 22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, Xia, and Braught, in view of Akkary et al. (USPN 6,247,121, herein Akkary).

33. As per Claim 9, Rotenberg teaches the microprocessor of claim 4, but fails to teach:

wherein the trace generator is configured to generate traces in response to instructions being retired.

While Rotenberg teaches that the instructions need to be decoded (Section 2.2) before the trace can be generated, it is not taught that they need to be retired beforehand (although it is suggested in Section 2.3, fill issues). Akkary teaches that instructions are not put into the trace buffers until they have been retired, to ensure that they executed correctly (Column 3, Lines 40-44). Rotenberg discusses in Section 2.3 that some traces are committed but never used, thus evicting a useful trace. By ensuring that the trace is correct, the odds that it will be useful is increased, as an incorrect (not actually executed) trace should probably not need to be used, as

branches tend to behave the same way (Section 1.1). Therefore, one of ordinary skill in the art at the time the invention was made would have waiting until an instruction was retired before considering adding it to the trace, in order to ensure accuracy of the trace and to prevent displacement of trace more likely to be reused.

34. As per Claim 22, Rotenberg teaches the computer system of claim 17, but fails to teach:

wherein the trace generator is configured to generate traces in response to instructions being retired.

While Rotenberg teaches that the instructions need to be decoded (Section 2.2) before the trace can be generated, it is not taught that they need to be retired beforehand (although it is suggested in Section 2.3, fill issues). Akkary teaches that instructions are not put into the trace buffers until they have been retired, to ensure that they executed correctly (Column 3, Lines 40-44). Rotenberg discusses in Section 2.3 that some traces are committed but never used, thus evicting a useful trace. By ensuring that the trace is correct, the odds that it will be useful is increased, as an incorrect (not actually executed) trace should probably not need to be used, as branches tend to behave the same way (Section 1.1). Therefore, one of ordinary skill in the art at the time the invention was made would have waiting until an instruction was retired before considering adding it to the trace, in order to ensure accuracy of the trace and to prevent displacement of trace more likely to be reused.

35. Claims 27-31 and 35 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, Xia, Braught, and Lange, further in view of Akkary.

36. As per Claim 27, Rotenberg teaches: A method, comprising:

beginning construction of a new trace (Section 2.2), but fails to teach:

receiving a retired instruction; and

determining if a previous trace under construction duplicates a trace in a trace cache and if the received instruction corresponds to a branch label; and

beginning construction of the trace in response to determining that a previous trace under construction duplicates a trace in a trace cache and that the received instruction corresponds to a branch label.

Rotenberg teaches starting a trace on a trace cache miss, which then causes the line-fill buffer to begin filling (Section 2.2, fourth paragraph). Rotenberg does not specifically teach that an instruction must be a branch to access the cache (despite requiring branch prediction bits). However, Xia teaches that an advantage of starting a trace only at predictable branch instructions (Section 5.5, first scheme) is that less cache space is required, in direct contrast to Rotenberg, which teaches tracing on all instructions, requiring significantly more space (second scheme). Given this advantage, one of ordinary skill in the art at the time the invention was made would have been motivated by the advantage of needing less space for the trace cache to implement traces only on branches.

However, this combination still fails to teach starting a trace on a label boundary. Braught teaches that in Assembly Language, most branches operate on labels, which the machine can only interpret when converted to an address (Page 3). In Braughts example on page 3, all branches are labels, making every branch a label boundary. When combined with Rotenberg's invention, this would mean that a trace would be generated with an instruction corresponding to a label boundary, as all the branches would go to labels, and even an address may be interpreted as a label. Given this knowledge, it would have been obvious to one of ordinary skill in the art that Rotenberg's invention begins traces when it encounters an instruction with a label boundary, as it only begins on branch instructions, which branch to labels.

While Rotenberg teaches that the instructions need to be decoded (Section 2.2) before the trace can be generated, it is not taught that they need to be retired beforehand (although it is suggested in Section 2.3, fill issues). Akkary teaches that instructions are not put into the trace buffers until they have been retired, to ensure that they executed correctly (Column 3, Lines 40-44). Rotenberg discusses in Section 2.3 that some traces are committed but never used, thus evicting a useful trace. By ensuring that the trace is correct, the odds that it will be useful is increased, as an incorrect (not actually executed) trace should probably not need to be used, as branches tend to behave the same way (Section 1.1). Therefore, one of ordinary skill in the art at the time the invention was made would have waiting until an instruction was retired before considering adding it to the trace, in order to ensure accuracy of the trace and to prevent displacement of trace more likely to be reused.

Rotenberg teaches a standard trace cache, with the potential for some duplication, when considering the alternative embodiment of path associativity disclosed in Section 2.3. And while Rotenberg does not discuss the issue of duplication, he does teach of disadvantages which occur when a trace cache miss occurs while servicing a previous trace cache miss, and that this situation needs to be avoided, otherwise features which increase cost or decrease performance must be implemented, and further teaches the disadvantages of a useless trace displacing a useful trace (Section 2.3, judicious trace selection). Therefore, Rotenberg teaches a system in which allows tracing of potential duplicate traces, and also a system which requires action when a miss occurs while servicing a miss. Lange teaches a system in which a cache is checked for a value while accessing memory, and if the identical value is found in the cache, the fetch to memory is aborted, freeing the memory to be used by a different operation (preventing a blocking load, Column 9, Lines 60-65) This is analogous to Rotenberg in the sense that Lange provides a motivation to look for duplicates in a cache to prevent a large delay and allow another operation to continue (analogous to finding a solution to what to do when encountering a miss while servicing a miss) by implementing a solution of searching for the duplicate, and not performing a long-latency operation on the duplicate value since it is not required (if the previous trace appears to be a duplicate of what is in the trace, there is no need to continue tracing, and the duplicate value can be discarded as taught by Lange (there is no need to perform an action on a duplicate value), preventing the line-fill buffer from blocking new traces, a long-latency event).

In an alternative viewpoint disclosed above, Rotenberg also teaches why it is disadvantageous to displace a useful trace with a useless trace (under the interpretation that a duplicate trace being stored in the trace cache is useless, as the original can always be accessed in its place), thus, with the potential for duplicate traces to exist with path associativity in Rotenberg's alternative embodiments, Rotenberg indicates in his judicial trace selection that storing a duplicate trace would be at best useless, and at worst displace a useful trace that may be used, providing a motivation to prevent these duplicate traces from entering the trace cache from the line fill buffer, further providing a motivation to find some method to handle these cases, which Lange provides, by simply discarding the duplicate value.

Given these advantages in both of these situations, one of ordinary skill in the art at the time the invention was made would have been motivated to combine the teachings of Lange's checking for duplicates and discarding of the duplicates to Rotenberg, in order to solve either or both of the problems of getting a miss while servicing a miss, and preventing eviction of a useful trace by a duplicate trace.

The fact that traces can only begin on branch labels means that the next trace cannot be started until an instruction represents a branch label, regardless of when the duplication is found and dealt with.

37. As per claim 28, Rotenberg teaches: The method of claim 27, further comprising continuing construction of an incomplete trace already in process in response to determining that the incomplete trace does not duplicate a trace in the trace cache

(Section 2.2).

38. As per Claim 29, Lange teaches: The method of claim 27, further comprising searching the trace cache for duplicate entries subsequent to completion of the previous trace under construction or the new trace (Column 5, Lines 5-10).

39. As per Claim 30, Rotenberg teaches: The method of claim 29, further comprising creating a new entry in the trace cache in response to no duplicate entry being identified (Section 2.2).

40. As per Claim 31, Lange teaches: The method of claim 29, further comprising discarding a trace in response to a duplicate entry being identified (Column 5, Lines 5-10).

41. As per Claim 35, Rotenberg teaches: A microprocessor comprising:

means for starting a new trace (Section 2.2), but fails to teach:

means for receiving a retired operation;

means for determining if a previous trace under construction duplicates a trace in a trace cache and if the received operation is a first operation at a branch label; and

means for starting a new trace in response to determining that a previous trace under construction duplicates a trace in trace cache and that the received operation is a first operation at a branch label.

Rotenberg teaches starting a trace on a trace cache miss, which then causes the line-fill buffer to begin filling (Section 2.2, fourth paragraph). Rotenberg does not specifically teach that an instruction must be a branch to access the cache (despite requiring branch prediction bits). However, Xia teaches that an advantage of starting a trace only at predictable branch instructions (Section 5.5, first scheme) is that less cache space is required, in direct contrast to Rotenberg, which teaches tracing on all instructions, requiring significantly more space (second scheme). Given this advantage, one of ordinary skill in the art at the time the invention was made would have been motivated by the advantage of needing less space for the trace cache to implement traces only on branches.

However, this combination still fails to teach starting a trace on a label boundary. Braught teaches that in Assembly Language, most branches operate on labels, which the machine can only interpret when converted to an address (Page 3). In Braughts example on page 3, all branches are labels, making every branch a label boundary. When combined with Rotenberg's invention, this would mean that a trace would be generated with an instruction corresponding to a label boundary, as all the branches would go to labels, and even an address may be interpreted as a label. Given this knowledge, it would have been obvious to one of ordinary skill in the art that Rotenberg's invention begins traces when it encounters an instruction with a label boundary, as it only begins on branch instructions, which branch to labels.

While Rotenberg teaches that the instructions need to be decoded (Section 2.2) before the trace can be generated, it is not taught that they need to be retired

beforehand (although it is suggested in Section 2.3, fill issues). Akkary teaches that instructions are not put into the trace buffers until they have been retired, to ensure that they executed correctly (Column 3, Lines 40-44). Rotenberg discusses in Section 2.3 that some traces are committed but never used, thus evicting a useful trace. By ensuring that the trace is correct, the odds that it will be useful is increased, as an incorrect (not actually executed) trace should probably not need to be used, as branches tend to behave the same way (Section 1.1). Therefore, one of ordinary skill in the art at the time the invention was made would have waiting until an instruction was retired before considering adding it to the trace, in order to ensure accuracy of the trace and to prevent displacement of trace more likely to be reused.

Rotenberg teaches a standard trace cache, with the potential for some duplication, when considering the alternative embodiment of path associativity disclosed in Section 2.3. And while Rotenberg does not discuss the issue of duplication, he does teach of disadvantages which occur when a trace cache miss occurs while servicing a previous trace cache miss, and that this situation needs to be avoided, otherwise features which increase cost or decrease performance must be implemented, and further teaches the disadvantages of a useless trace displacing a useful trace (Section 2.3, judicious trace selection). Therefore, Rotenberg teaches a system in which allows tracing of potential duplicate traces, and also a system which requires action when a miss occurs while servicing a miss. Lange teaches a system in which a cache is checked for a value while accessing memory, and if the identical value is found in the cache, the fetch to memory is aborted, freeing the memory to be used by a different

operation (preventing a blocking load, Column 9, Lines 60-65) This is analogous to Rotenberg in the sense that Lange provides a motivation to look for duplicates in a cache to prevent a large delay and allow another operation to continue (analogous to finding a solution to what to do when encountering a miss while servicing a miss) by implementing a solution of searching for the duplicate, and not performing a long-latency operation on the duplicate value since it is not required (if the previous trace appears to be a duplicate of what is in the trace, there is no need to continue tracing, and the duplicate value can be discarded as taught by Lange (there is no need to perform an action on a duplicate value), preventing the line-fill buffer from blocking new traces, a long-latency event).

In an alternative viewpoint disclosed above, Rotenberg also teaches why it is disadvantageous to displace a useful trace with a useless trace (under the interpretation that a duplicate trace being stored in the trace cache is useless, as the original can always be accessed in its place), thus, with the potential for duplicate traces to exist with path associativity in Rotenberg's alternative embodiments, Rotenberg indicates in his judicial trace selection that storing a duplicate trace would be at best useless, and at worst displace a useful trace that may be used, providing a motivation to prevent these duplicate traces from entering the trace cache from the line fill buffer, further providing a motivation to find some method to handle these cases, which Lange provides, by simply discarding the duplicate value.

Given these advantages in both of these situations, one of ordinary skill in the art at the time the invention was made would have been motivated to combine the

teachings of Lange's checking for duplicates and discarding of the duplicates to Rotenberg, in order to solve either or both of the problems of getting a miss while servicing a miss, and preventing eviction of a useful trace by a duplicate trace.

The fact that traces can only begin on branch labels means that the next trace cannot be started until an instruction represents a branch label, irregardless of when the duplication is found and dealt with.

42. Claims 32-34 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, in view of Xia.

43. As per Claim 32, Rotenberg teaches: A method, comprising:

fetching instructions from an instruction cache (Section 2.1, paragraphs 1-3);

continuing to fetch instructions from the instruction cache until a branch target address is generated (Section 2.2, where it is said that "on a trace cache miss, fetching proceeds normally from the instruction cache);

in response to a branch target address being generated, searching trace cache for an entry corresponding to the branch target address (Section 2.2, third paragraph), but fails to teach:

without searching a trace cache;

Rotenberg teaches starting a trace on a trace cache miss, which then causes the line-fill buffer to begin filling (Section 2.2, fourth paragraph). Rotenberg does not teach not searching the trace cache until a branch target address is generated, and in fact

teaches that the trace cache is searched on every instruction. However, Xia teaches that an advantage of starting a trace only at predictable branch instructions (Section 5.5, first scheme) is that less cache space is required, in direct contrast to Rotenberg, which teaches tracing on all instructions, requiring significantly more space (second scheme). Given this advantage of using less cache space, one of ordinary skill in the art at the time the invention was made would have been motivated by the advantage of needing less space for the trace cache to implement traces only on branches. An effect of only having traces start on branches is that it is then inherent that a non-branch instruction can never be the start of a trace, therefore there would be no reason to search the trace cache for a non-branch instruction, which has further potential advantages such as power saving and potential critical path reduction, as one of ordinary skill in the art would recognize.

44. As per Claim 33, Rotenberg teaches: The method of claim 32, further comprising continuing to fetch instructions from the instruction cache in response to no entry being identified in the trace cache corresponding to the branch target address (Section 2.2, where it is said that "on a trace cache miss, fetching proceeds normally from the instruction cache).

45. As per Claim 34, Rotenberg teaches: The method of claim 32, further comprising fetching one or more traces from the trace cache in response to an entry being identified in the trace cache corresponding to the branch target address (Section 2.2,

where it is said that "on a trace cache hit, an entire trace of instructions is fed into the instruction latch, bypassing the instruction cache).

Response to Arguments

46. Regarding Applicants arguments for claims 1 and 14 (the 102 rejections), Applicant has argued that because Rotenberg checks the trace cache for every fetch address, that Rotenberg cannot read on the limitation that the prefetch unit is configured to "not check the trace cache for a match until the branch prediction unit outputs the predicted target address". However, Applicant is twisting the meaning of the words in the claim, and it is extremely apparent that there is absolutely no way that any machine can possibly function differently than what Applicant has claimed, which is why the specific limitations recited in the claims are inherent. The claims state that the prefetch unit is configured to check the trace cache for a match for the predicted target address in response to the branch prediction unit outputting the predicted target address, and then that the prefetch unit is configured to not check the trace cache for a match until the branch prediction unit outputs the predicted target address. When looking at the entire claim in context, it can be seen that the claim is referring to searching the cache for a match for the predicted target address, and that it cannot search the cache for the predicted target address until the predicted target address is generated. Thus, the fact that it searches for the fetch address on every cycle is completely immaterial to this rejection, because the fetch address is not the predicted target address. It is absolutely 100% impossible to search for something that has not been generated yet. If Applicant

is somehow claiming that this is the case, than there are serious enablement issues with the claims. Therefore, the rejection under 102 stands.

47. Regarding the 103 rejections to the claims, Applicant has first argued over the use of Xia, saying that Xia shows that trace lines can only start from predictable branches, which does not include all types of branches, nor that these instructions are associated with a branch label (which is why Examiner combined Braught into the reference). However, the claims do not in any way state or even suggest that all branches have labels. All the claim discloses is that traces are started on branch labels. If even one out of a thousand branches had a label, it would be sufficient to read on the claim, and by Applicant arguing that Xia needs to teach that all branches have labels in order to read on the claim is completely unfounded in the claims, and is not even remotely suggested at any point, thus the argument is completely moot.

Further, Applicant has argued that Braught fails to teach that all branches have labels, thus somehow does not read on the claim, and has argued that Examiner has misquoted Braught. However, when Braught states "Most of the remaining Assembly Language Instructions operate on Labels", that is equivalent to most branches operating on labels, because Braught was referring to ALU operations and shifts above, and branches are not an ALU operation and are not a shift operation, thus it is pretty clear that branches fall into the "remaining" instructions. Additionally, every single branch shown in Braught uses a label, and Examiner feels it is absolutely crystal clear that Braught teaches using branch labels, and that all of the branches show in Braught do

use labels (additionally, anyone who has programmed in Assembly would recognize that labels are used for branches). Given that Braught teaches that branches use labels, and that every single example of a branch in Braught uses a label, Examiner feels that Braught easily supports the teachings of branches having labels.

48. Applicant has further argued over the "receiving a retired instruction" limitation. While the first arguments have been discussed in the previous actions, the new argument regarding this limitation is presented on Page 14, where Applicant has argued that Rotenberg, while teaching the goal that there is an advantage to storing correctly executed instructions, that Rotenberg teaches filling the buffer as instructions are fetched from the instruction cache, thus teach away from filling it with retired instructions. However, as Examiner pointed out, In section 2.3, Fill Issues, Rotenberg indicates that instead of filling the cache with speculative traces (on fetch), that it may be done by waiting for the branch outcomes before committing a trace (which is clearly not on fetching). Applicant has also argued that the Examiners pointing to the "judicious trace selection" section does not suggest waiting until an instruction was retired before adding it to the trace, because Rotenberg suggests that one possible solution is to add a buffer. However, Applicant has ignored the rejection in the context of which it was given, and is misreading Rotenberg in addition to this. Rotenberg suggests one possible solution, it is never stated that it is the only or even the best way to deal with this problem, but as Examiner pointed out in the rejection, the key feature is that there is clearly a very damaging effect on the trace cache for storing non-useful traces, and

given that Rotenberg not only suggests that there is a motivation and reason to wait for the correct result for the instruction (by the branch outcome being resolved), but that Akkary clearly states that a retired instruction is guaranteed to have executed, and executed correctly, it would have been very obvious for one of skill in the art to come to the conclusion that if Akkary teaches a retired instruction will have been correct, and Rotenberg teaches that there is not only an advantage to using correct instructions, but that incorrect instructions may cause a performance penalty in the trace cache, that there is more than enough motivation and suggestion to combine the references. Additionally, Applicant has argued that the "fill issues" section teaches about whether to fill the cache with speculative instructions, and not whether to add individual instructions to the cache or start construction of a trace. However, a trace cannot be started without adding in a trace, thus determining what instructions to add to the trace is directly tied into starting a trace, and as stated above in this paragraph, Examiner feels that this section does teach when to add instructions to the trace.

49. Applicant has next argued over the duplication limitation. Applicant has asserted that Examiner has contradicted himself saying that Rotenberg has a system which can potentially have duplicate traces, and that there is no support for saying such in Rotenberg. Firstly, Examiner does not believe he is contradicting himself in saying that Rotenberg does not discuss the issue of duplication, yet teaches a system where duplicates can exist. Rotenberg does not talk about duplication, and does not suggest that the issue can occur, and in fact, in the main embodiment, it is impossible for a

duplicate to exist. However, Rotenberg does offer an alternative embodiment which does create the potential for duplicate traces to exist, in the "path associativity" alternative embodiment, which Applicant appears to have overlooked. In this alternate embodiment, Rotenberg teaches that it may be advantageous to store multiple paths emanating from a given address, as it is possible that a branch in the middle of the trace will create two possible outcomes that should be saved. Now, it should be very apparent from this teaching that the only possible way to implement this is to perform a trace whenever a trace would normally be started, even on a cache hit, because the path may diverge from the stored trace, and be unique, and there is absolutely no way to know until the trace is complete. However, if it is not unique, then it would end up being a duplicate of the trace that was hit in the cache, and it would be wasteful to allow this trace to enter the cache (where it may overwrite a different trace based on the implementation of the cache). Thus Rotenberg discusses an alternative embodiment in which duplicates could exist, even though he does not explicitly mention it or say how to deal with it, but one of ordinary skill in the art would be able to understand this by looking at the alternative embodiment.

50. Applicant has further argued that Lange's teachings have "absolutely nothing to do with the limitations of the present invention", and has said that "a method for reducing fetches from memory by first checking a cache is not analogous to a method for reducing the amount of data that needs to be loaded into a trace cache by checking the contents of the trace cache itself". Examiner strongly disagrees, and believes that

Applicant's own arguments strongly show the connection. Reducing fetches from memory would clearly result in less data being loaded into the cache, because a fetch to memory takes data from main memory and puts it in the cache. This is exactly the same as reducing the amount of data that needs to be loaded into the trace cache. They are completely analogous, because they both relate to reducing the amount of data that needs to be written into a cache. As for Applicants remaining arguments against Lange, Examiner has already addressed the duplication issue above.

51. Regarding Claim 32, Applicant has argued that Xia checks the trace and instruction cache in every instruction, thus that it would not be obvious to not search the trace cache on a non-branch instruction. However, Examiner believes Applicant is misinterpreting the reference, and Examiner believes that the section of Xia that Applicant is referring to is retrieving instructions for a trace that has already been recognized as in progress, and is not searching for the start of a trace. As stated in Xia, at the top of page 4, it is explicitly stated that "A trace line won't start from a non-branch instruction". Examiner finds it hard to believe that this would lead one of ordinary skill in the art to check a trace cache for the start of a trace, if it is guaranteed that the instruction cannot possibly be the start of a trace line. It would be nonsensical to search for something that is guaranteed to not possibly be there, and a waste of time and energy to do so. The advantages that Examiner indicated, power saving and critical path reduction, while not explicitly stated in the reference, are obvious advantages one of ordinary skill in the art would recognize, searching a cache takes time and energy,

and if there is no possible way that searching the cache can benefit the user, there is absolutely no reason to do so.

Conclusion

52. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Robert E. Fennema whose telephone number is (571) 272-2748. The examiner can normally be reached on Monday-Friday, 8:45-6:15.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Robert E Fennema
Examiner
Art Unit 2183

RF

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100